**THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE ARE CLAIMED ARE DEFINED AS FOLLOWS:**

1. A computer program product comprising a computer usable medium tangibly embodying computer readable program code for defining code to provide a locking mechanism for self-modifying code in a multi-thread environment, the self-modifying code comprising helper code callable to modify instructions in a defined block of the self-modifying code, said computer program product comprising:

computer readable program code means for defining an atomic compare and exchange instruction in the locking mechanism,

the defined atomic compare and exchange instruction for carrying out a comparison of an unreserved lock value with a first instruction in the defined block of self-modifying code,

the defined atomic compare and exchange instruction for exchanging the first instruction in the defined block of self-modifying code with a self-loop instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code;

computer readable program code means for defining code to return execution to the first instruction in the defined block of self-modifying code where the comparison indicates that the unreserved lock value does not match the first instruction in the defined block of self-modifying code; and

computer readable program code means for defining code to permit the remainder of the helper code to be executed to carry out modifications in the defined block of self-modifying code, including as a last step an atomic store to replace the self-loop instruction with a modified instruction, where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code.

2. The computer program product of claim 1 further comprising computer readable program code means for defining the first instruction in the defined block of self-modifying code to be a call instruction to the helper code and for defining the unreserved lock value to be calculated in the helper code based on a return call instruction address passed to the helper code.

3. The computer program product of claim 1 further comprising computer readable program code means for defining the first instruction in the defined block of self-modifying code to be a call instruction to the helper code and for storing the unreserved lock value as a binary encoding of the call instruction available to the helper code.

4. The computer program product of claim 2 in which the helper code is loaded at a non-boundary position in memory.

5. The computer program product of claim 1 further comprising computer readable program code means for defining the first instruction in the defined block of self-modifying code

to be an illegal instruction to interact with a defined trap handler to pass control to the helper code, and for defining the unreserved lock value to be the binary encoding of the illegal instruction.

6. The computer program product of claim 1 in which the helper code replaces unresolved references in the defined block of self-modifying code.

7. A computer program product comprising a computer usable medium tangibly embodying computer readable program code means for defining code to provide a locking mechanism for self-modifying code in a multi-thread environment, the self-modifying code comprising helper code callable to resolve unresolved references in a defined block of the self-modifying code, the helper code loaded at a non-boundary position in memory, said computer program product comprising:

computer readable program code means for defining an atomic compare and exchange instruction in the locking mechanism,

the defined atomic compare and exchange instruction for carrying out a comparison of an unreserved lock value with a first instruction in the defined block of self-modifying code,

the defined atomic compare and exchange instruction for exchanging the first instruction in the defined block of self-modifying code with a self-loop instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code;

computer readable program code means for defining code to return execution to the first instruction in the defined block of self-modifying code where the comparison indicates that the unreserved lock value does not match the first instruction in the defined block of self-modifying code;

computer readable program code means for defining code to permit the remainder of the helper code to be executed to resolve references in the defined block of self-modifying code, including as a last step an atomic store to replace the self-loop instruction with a modified instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code; and

computer readable program code means for defining the first instruction in the defined block of self-modifying code to be a call instruction to the helper code and the unreserved lock value is calculated in the helper code based on a return address passed to the helper code.

8. A method for locking self-modifying code in a multi-thread environment, the self-modifying code comprising helper code callable to modify instructions in a defined block of the self-modifying code, the method comprising:

defining an atomic compare and exchange instruction in the locking mechanism,

the defined atomic compare and exchange instruction carrying out a comparison of an unreserved lock value with a first instruction in the defined block of self-modifying code,

the defined atomic compare and exchange instruction exchanging the first instruction in the defined block of self-modifying code with a self-loop instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code;

defining code to return execution to the first instruction in the defined block of self-modifying code where the comparison indicates that the unreserved lock value does not match the first instruction in the defined block of self-modifying code; and

defining code to permit the remainder of the helper code to be executed to carry out modifications in the defined block of self-modifying code, including as a last step an atomic store to replace the self-loop instruction with a modified instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code.

9. The method of claim 8 further comprising generating the first instruction in the defined block of self-modifying code to be a call instruction to the helper code and of defining code for calculating the unreserved lock value in the helper code based on a return address passed to the helper code.

10. The method of claim 8 further comprising generating the first instruction in the defined block of self-modifying code to be a call instruction to the helper code and of defining code for storing the unreserved lock value as a binary encoding of the call instruction available to the helper code.

11. The method of claim 9 further comprising the step of defining code for loading the helper code at a non-boundary position in memory.

12. The method of claim 8 further comprising the steps of generating the first instruction in the defined block of self-modifying code to be an illegal instruction to interact with a defined trap handler to pass control to the helper code, and of defining code for setting the unreserved lock value to be the binary encoding of the illegal instruction.

13. The method of claim 8 in which the helper code replaces unresolved references in the defined block of self-modifying code.

14. A locking mechanism for self-modifying code in a multi-thread computer system, the self-modifying code comprising helper code callable to modify instructions in a defined block of the self-modifying code, the locking mechanism comprising:

an atomic compare and exchange instruction,

the atomic compare and exchange instruction for carrying out a comparison of an unreserved lock value with a first instruction in the defined block of self-modifying code,

the atomic compare and exchange instruction for exchanging the first instruction in the defined block of self-modifying code with a self-loop instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code;

the locking mechanism including code defined to return execution to the first instruction in the defined block of self-modifying code where the comparison indicates that the unreserved lock value does not match the first instruction in the defined block of self-modifying code;

the locking mechanism including code defined to permit the remainder of the helper code to be executed to carry out modifications in the defined block of self-modifying code, including as a last step an atomic store to replace the self-loop instruction with a modified instruction where the comparison indicates that the unreserved lock value matches the first instruction in the defined block of self-modifying code.

15. The locking mechanism of claim 14 in which the first instruction in the defined block of self-modifying code is a call instruction to the helper code and the unreserved lock value is calculated in the helper code based on a return address passed to the helper code.

16. The locking mechanism of claim 14 in which the first instruction in the defined block of self-modifying code is a call instruction to the helper code and the unreserved lock value is a stored binary encoding of the call instruction available to the helper code.

17. The locking mechanism of claim 15 in which the helper code is stored at a non-boundary position in memory.

18. The locking mechanism of claim 14 in which the first instruction in the defined block of self-modifying code is an illegal instruction defined to interact with a defined trap handler to pass control to the helper code, and in which the unreserved lock value is the binary encoding of the illegal instruction.

19. The locking mechanism of claim 14 in which the helper code replaces unresolved references in the defined block of self-modifying code.

20. A method for generating executable computer code to define a locking mechanism for runtime resolution of unresolved references in a specified block of executable code in a multithread environment, the method comprising:

a) inserting a call instruction at a first instruction address in the specified block of executable code, the call instruction branching to a block of helper code;

b) defining the lock mechanism in the helper code by:

i) including instructions in the helper code to calculate a binary encoding for the call instruction at the first instruction address,

ii) including an atomic compare and exchange instruction in the helper code, said included instruction having arguments defined to be a calculated binary encoding for the call instruction, the calculated binary encoding for a self loop instruction, and the first instruction address,

iii) including a branch to the first instruction address in the specified block of executable code, the branch being taken when the included atomic compare and exchange instruction identifies that the calculated binary encoding for the call instruction does not match contents at the first instruction address;

c) defining instructions in the helper code for the resolution of unresolved references in the specified block of executable code, the last such instruction being defined to be and an atomic store instruction to replace the instruction at the first instruction address.

21. A computer program product comprising a computer usable medium tangibly embodying computer readable program code means for carrying out the method of claim 20.